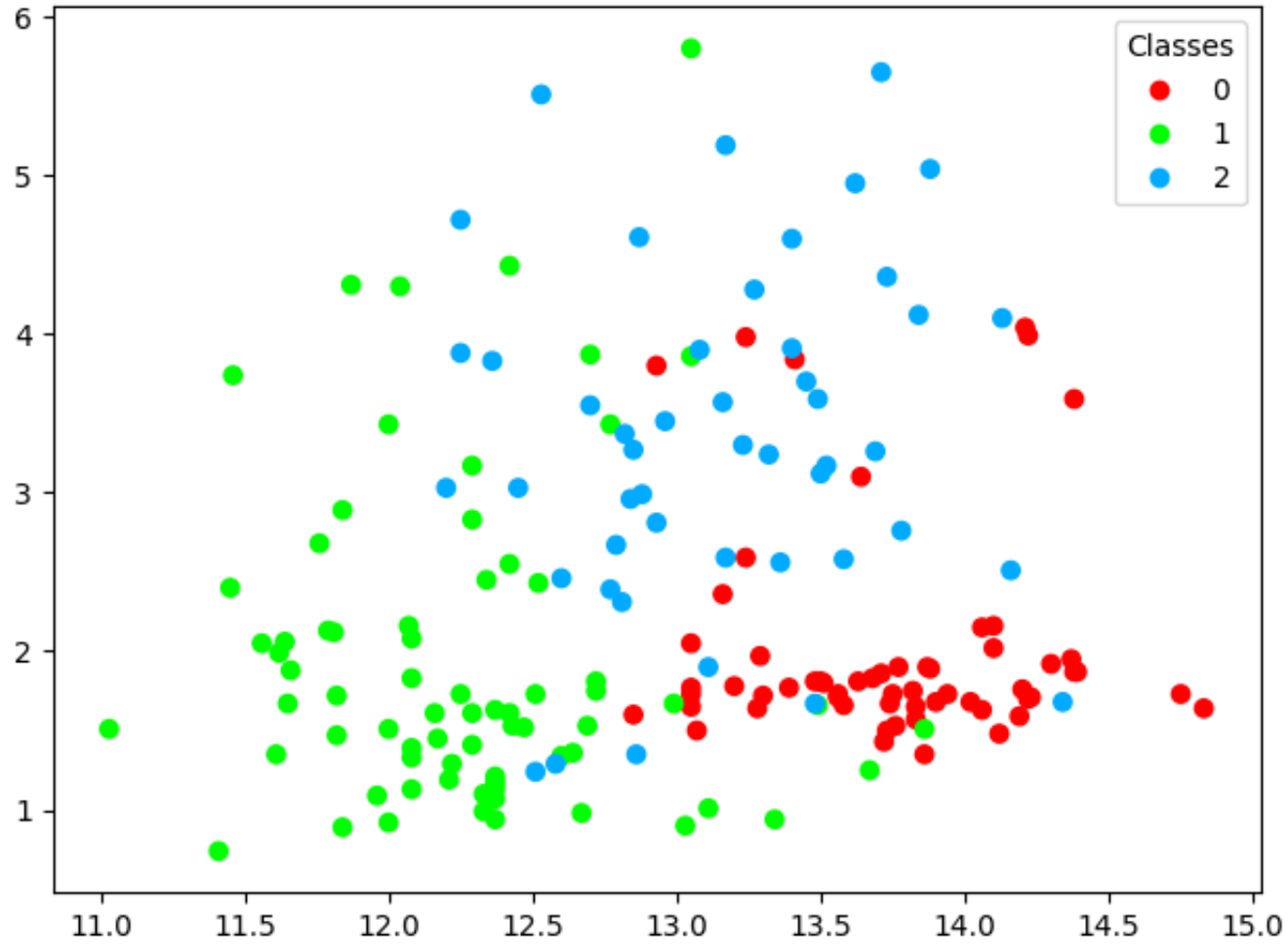
Homework 1-Bis

Student: Miro Filomeno Davide 256870

Dataset

1,2,3) In these analysis was used the wine dataset.This dataset is characterized of 178 elements ,each with 13 features, and divided in 3 classes.A graphical representation of the dataset is the following:



In this representation was shown only the first 2 features that were also used to the analysis. The datasets was splits in 3 parts (train,validation and test sets)in the proportion 5:2:3 with the following code:

```
X_train,X_t,Y_train,Y_t= train_test_split(X.data,X.target,test_size=0.5,random_sta
te=1)
X_validation,X_test,Y_validation,Y_test = train_test_split(X_t,Y_t,test_size=0.6)
```

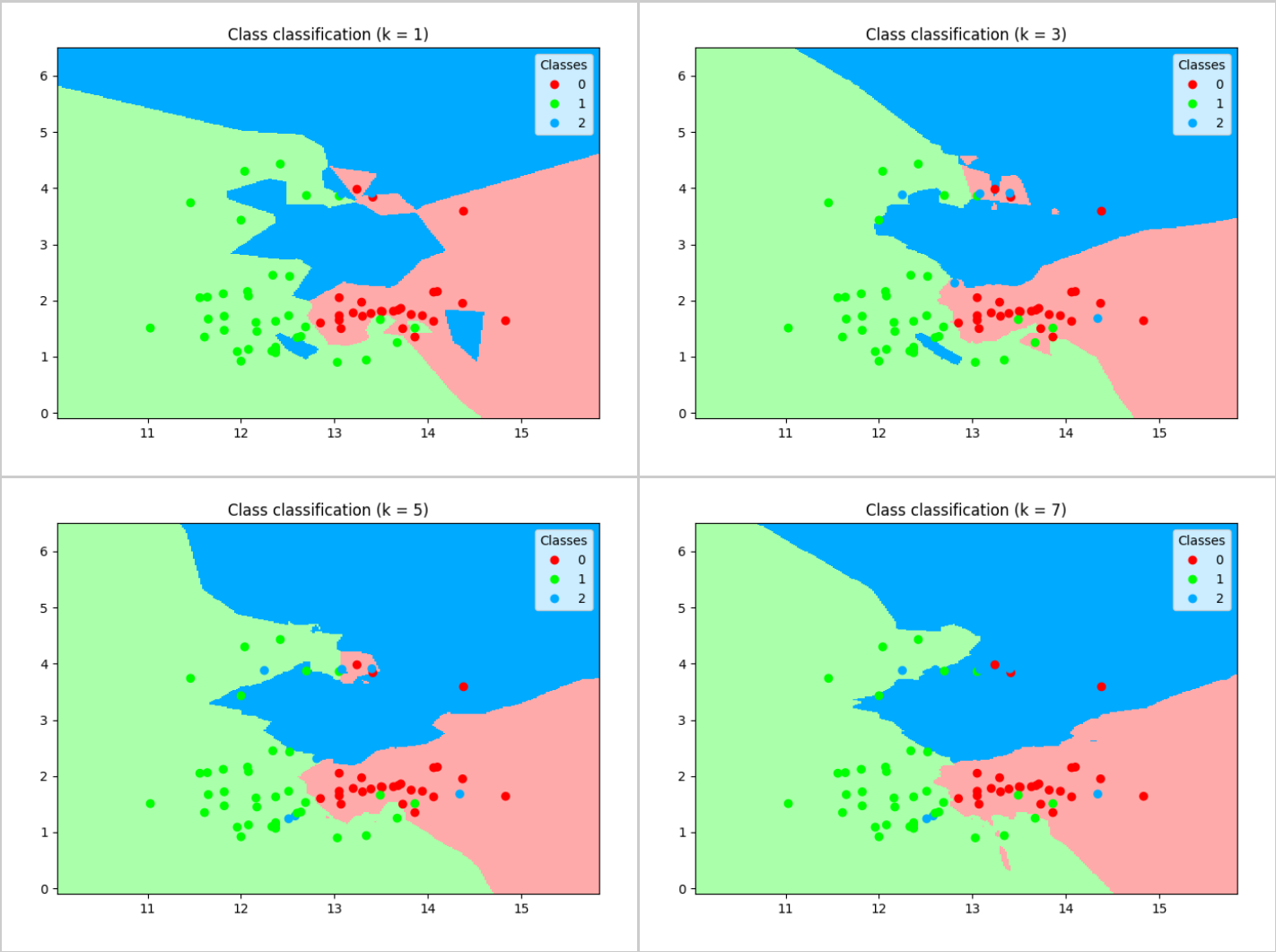
The paramiter **random_state=1** is set to get the same split with each execution

K-Nearest Neighbour

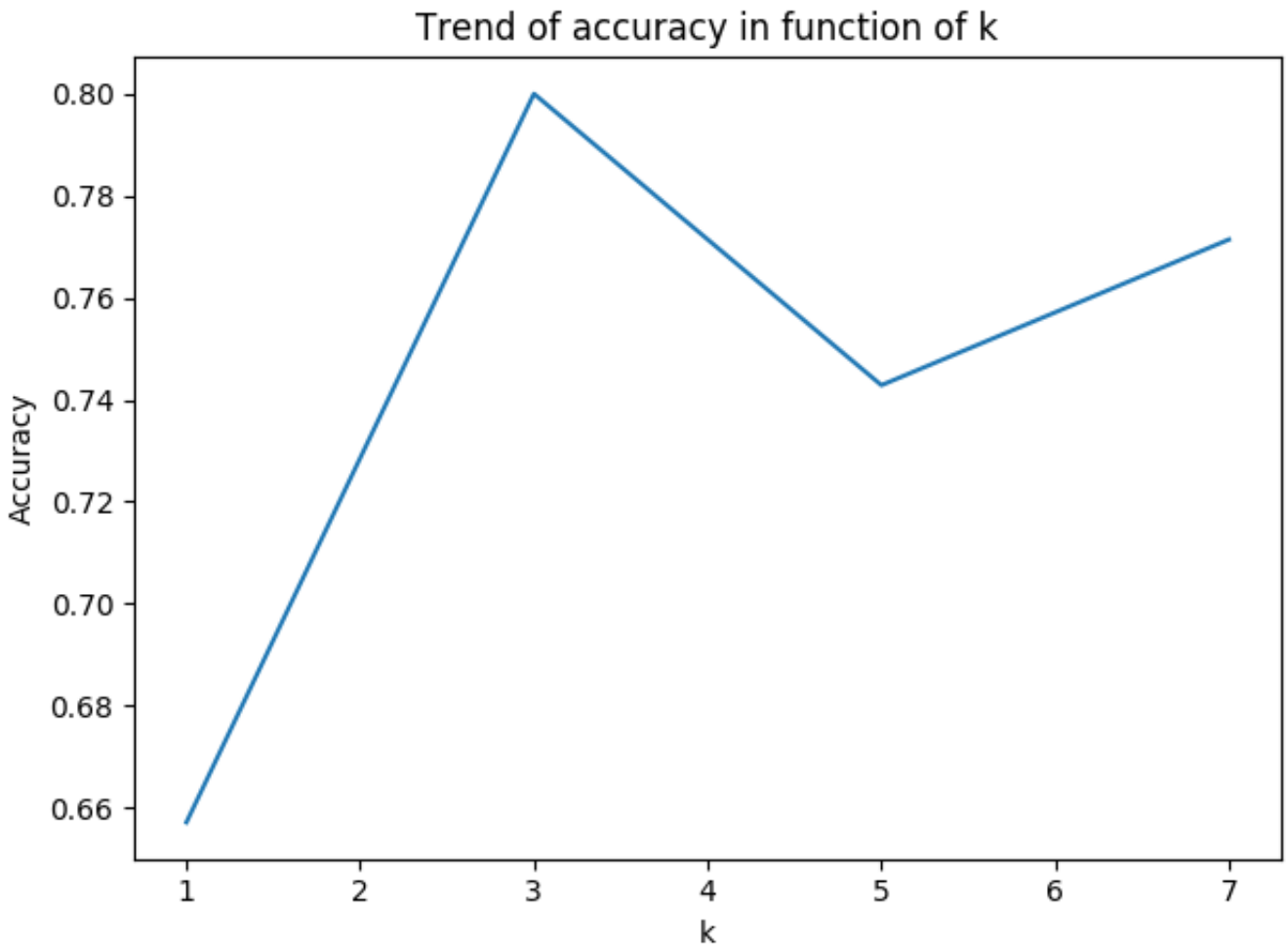
4)This method was applied for k=[1,3,5,7] and below is shown the code for k = 1:

```
knn_1 =knn.KNeighborsClassifier(1)
knn_1.fit(X_train[:,:2],Y_train)
Y_predict = knn_1.predict(X_validation[:,:2])
score1 = mc.accuracy_score(Y_validation,Y_predict)
```

For other paramiters the code is the same.A graphical representation of these models:



5)In the graph below is shown the trend of the accuracy scores in function of k using validation set.



6)The boundaries change because change the number of nearest to be take in consideration. A lower k makes the model sensible to outliers and can lead to misclassification for not well defined class distributions. An higher value of k lead to take in consideration points of different classes in classification process. You can see this phenomeno in the charts of k=3,5,7.In the char k=3 you can see a red area with some red points between the big areas blue and green.You can see that the increasing k this small region gradually became blue and this can lead to misclassification.

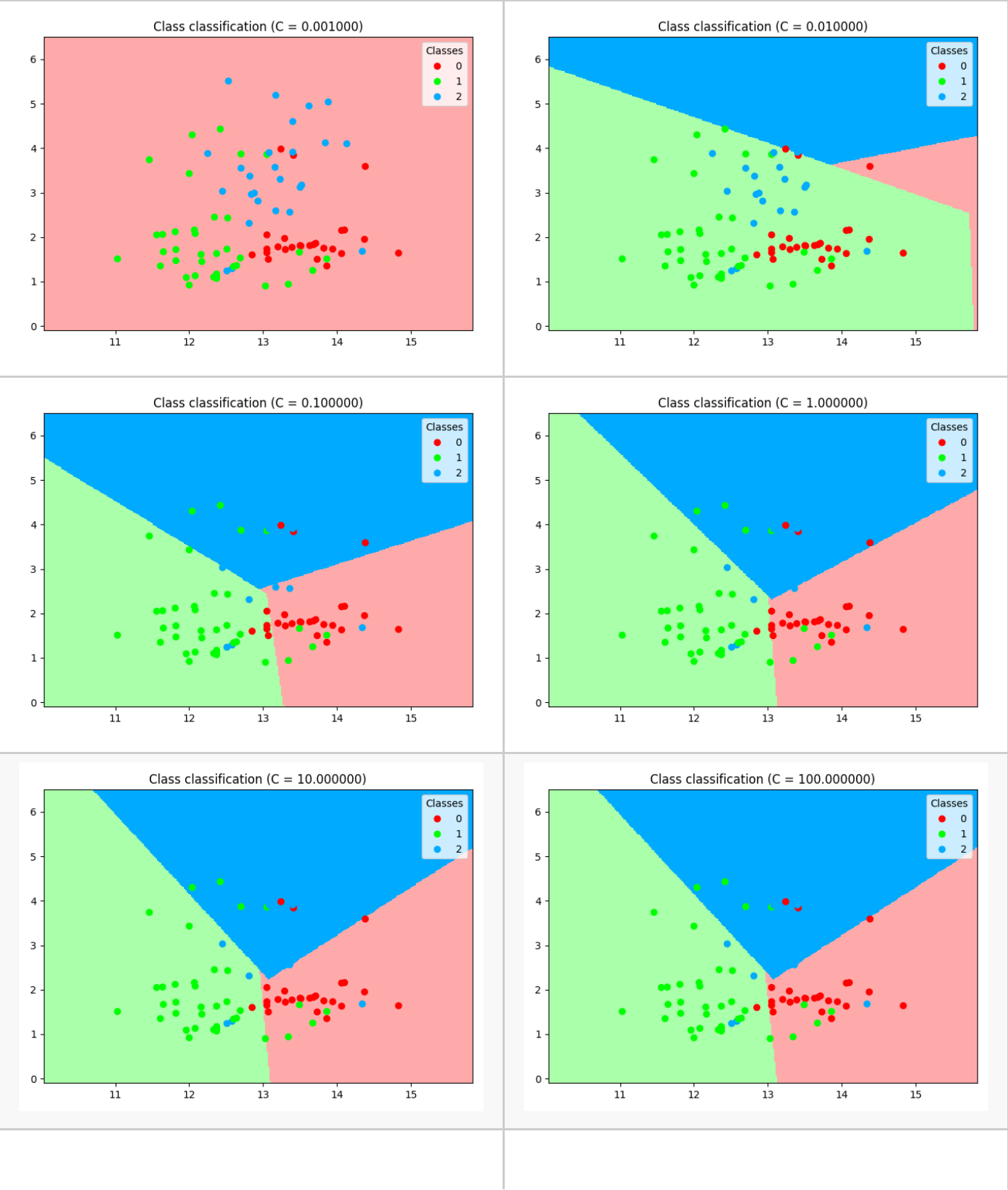
7)As shown in the chart,at the end of validation phase the best k is 3.After selecting the paramiter,the resulting model is a KNN with k=3 fitted over Xtrain+Xvalidation.The accuracy score of the resulting model,computed with Xtest,is about **0.76**.

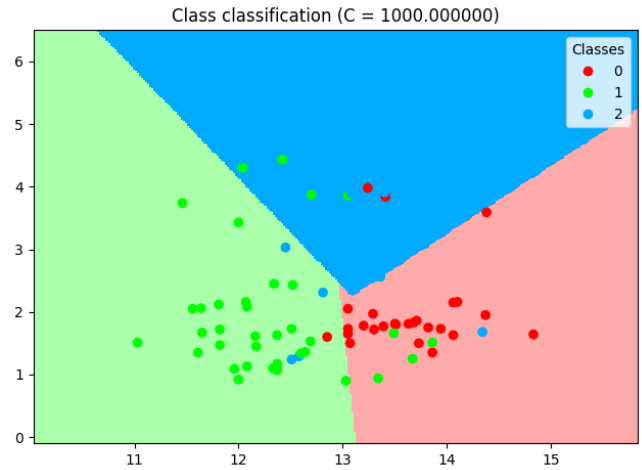
Linear SVM

8)This algorithm is applied for C = [0.001,0.01,0.1,1,10,100,1000] using Xtrain for fitting and Xvalidation to compute the accuracy and select the best C.This is the code used to do this:

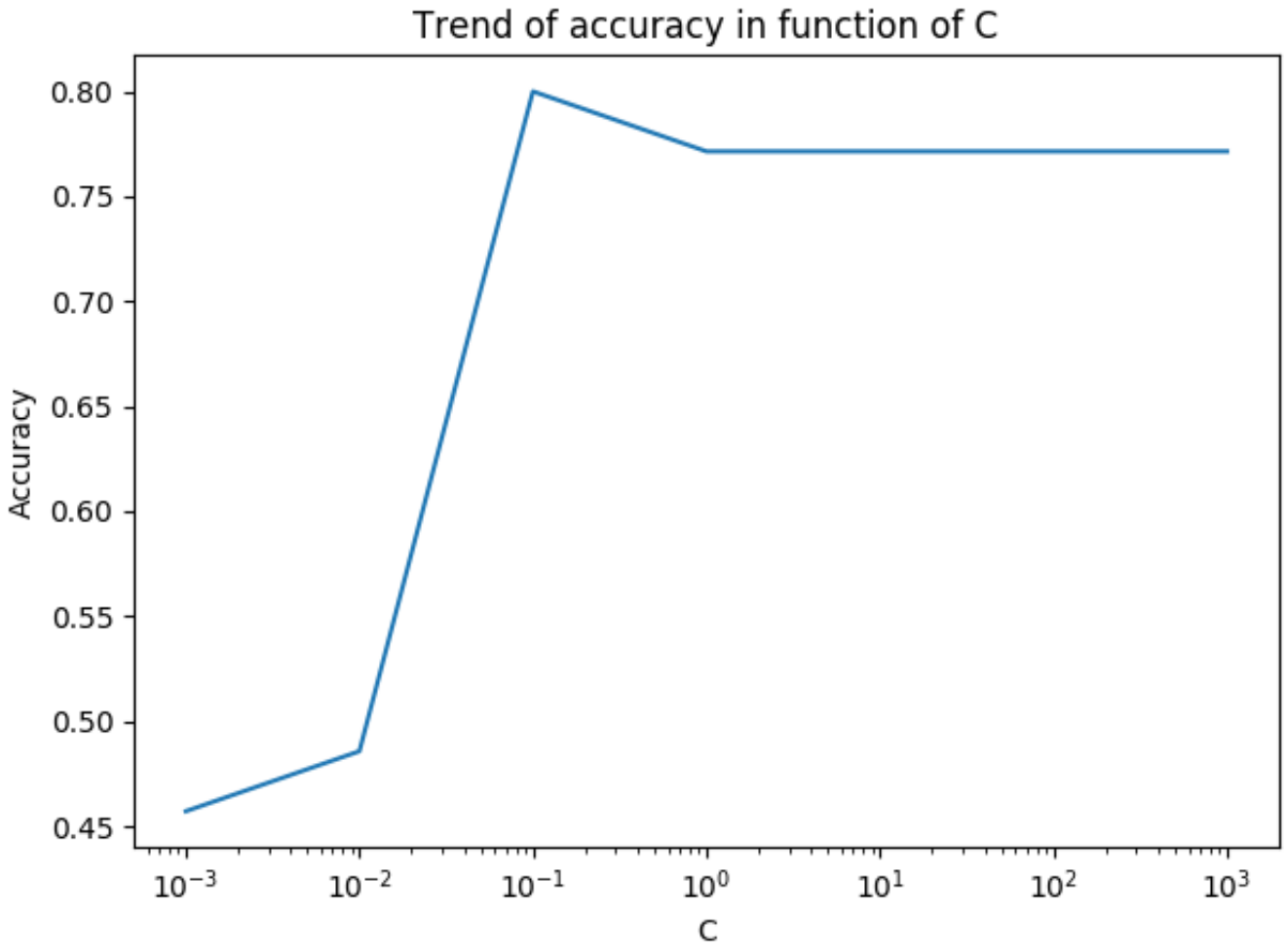
```
svm_001 = svm.SVC(kernel='linear',C=0.001)
svm_001.fit(X_train[:,:2],Y_train)
Y_predict = svm_001.predict(X_validation[:,:2])
score_001 = mc.accuracy_score(Y_validation,Y_predict)
```

For other paramiters the code is the same.A graphical representation of these models:





9)In the graph below is shown the trend of the accuracy scores in function of C using validation set.



10) The boundaries changement depends on C paramiter than for smaller values allow larger margin that avoid overfitting allowing some misclassificated points inside them and an higher C generate smaller margin with less misclassificated training data but at cost of a not well generalized model. In the figures above is visible this phenomeno, in the first picture there are the boundaries of the model with C=0.001,a very small value and It's visible a very big margin that define only the region of class 2 and allow misclassification of the other elements of the remained 2 classes.Increasing C the margins become smaller and the model more accurate with prediction of training data,but starting from C=1 the model became less accurate in prediction of validation element due to overfitting.Infact for C=0.1 we have an accuracy on the training data equal to **0.74** but a better score in the validation phase **0.8** instead of C = 1 with a score of **0.79** in training data prediction and a **0.77** in validation.

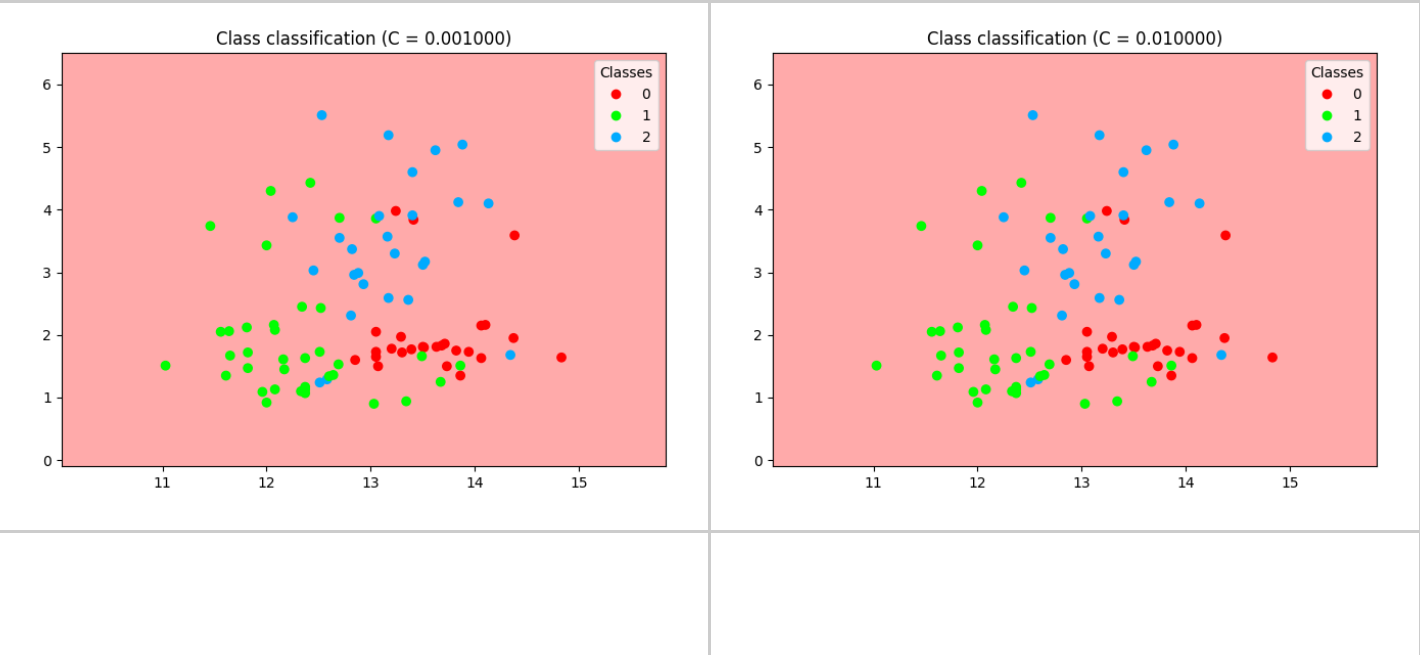
11) According to the accuracy score over the validation set,the C selected is C=0.1,the resulting model is given by using Xtrain+Xvalidation as training set.The resulting model provides an accuracy score over the test set about **0.796**

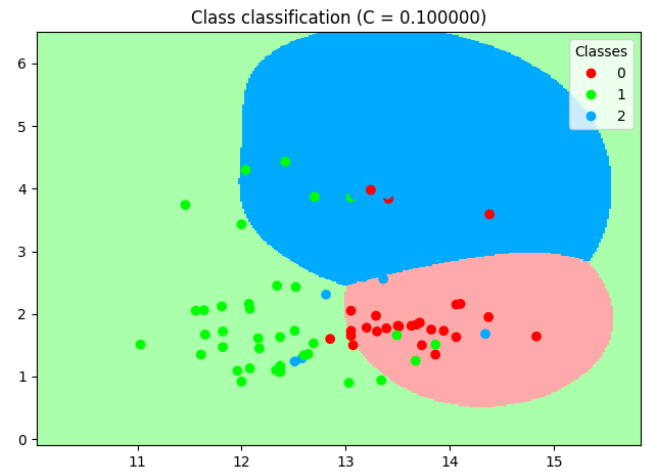
RBF Kernel

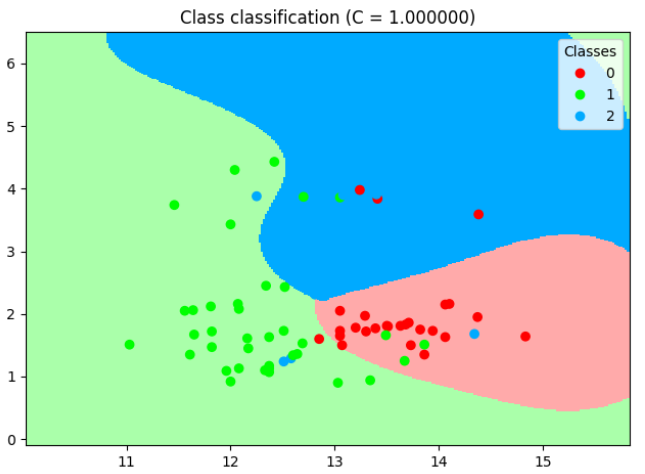
12) With an accurate observation of class distribution is clear that don't exist some linear boundaries that allow a perfect separation of the classes.It's a good idea to use the kernel function rbf. The code used is:

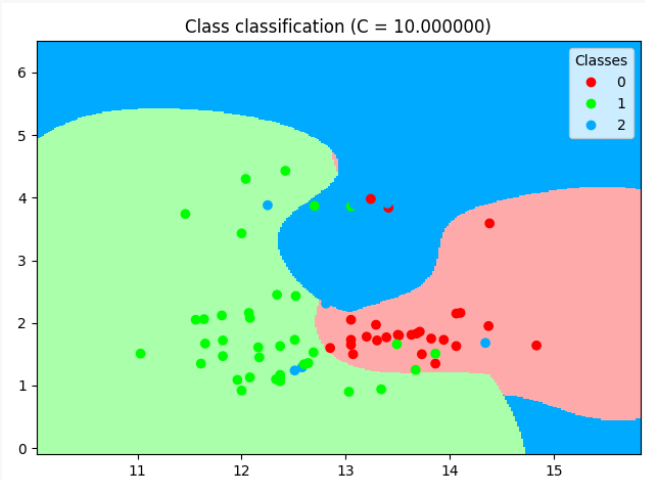
```
svm_001 = svm.SVC(kernel='rbf',C=0.001,gamma='auto')
svm_001.fit(X_train[:,:2],Y_train)
Y_predict = svm_001.predict(X_validation[:,:2])
score_001 = mc.accuracy_score(Y_validation,Y_predict)
```

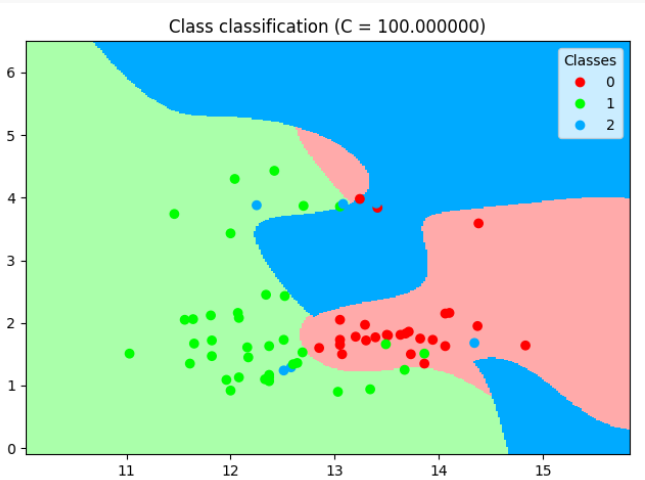
This analysis was performed for C = [0.001,0.01,0.1,1,10,100,1000] and gamma ='auto' for all C,that according to sklearn documentation means that gamma = 1/n_features and so gamma = 0.5. These are the the graphical representations of the new boundaries:

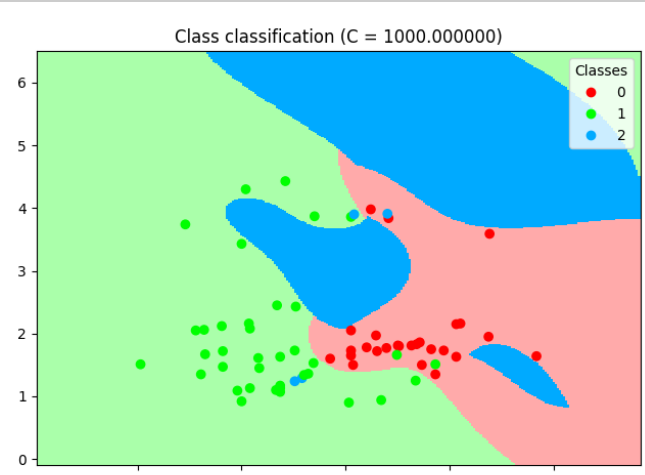



Class classification (C = 0.100000)
Classes
0
1
2
0
1
2
3
4
5
6
11
12
13
14
15

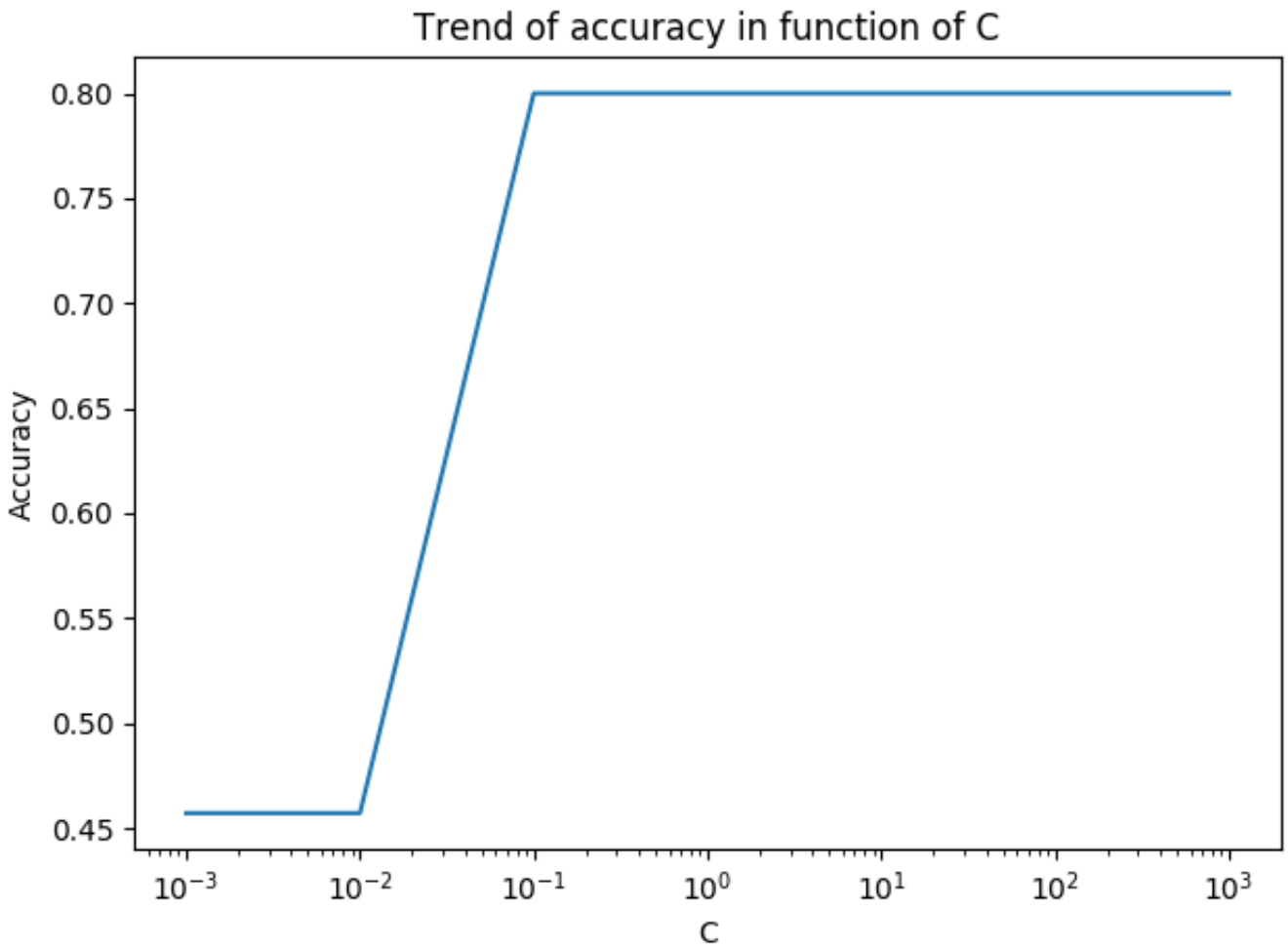

Class classification (C = 1.000000)
Classes
0
1
2
0
1
2
3
4
5
6
11
12
13
14
15


Class classification (C = 10.000000)
Classes
0
1
2
0
1
2
3
4
5
6
11
12
13
14
15


Class classification (C = 100.000000)
Classes
0
1
2
0
1
2
3
4
5
6
11
12
13
14
15


Class classification (C = 1000.000000)
Classes
0
1
2
0
1
2
3
4
5
6
11
12
13
14
15

The chart of the accuracy over the validation set in function of C is:



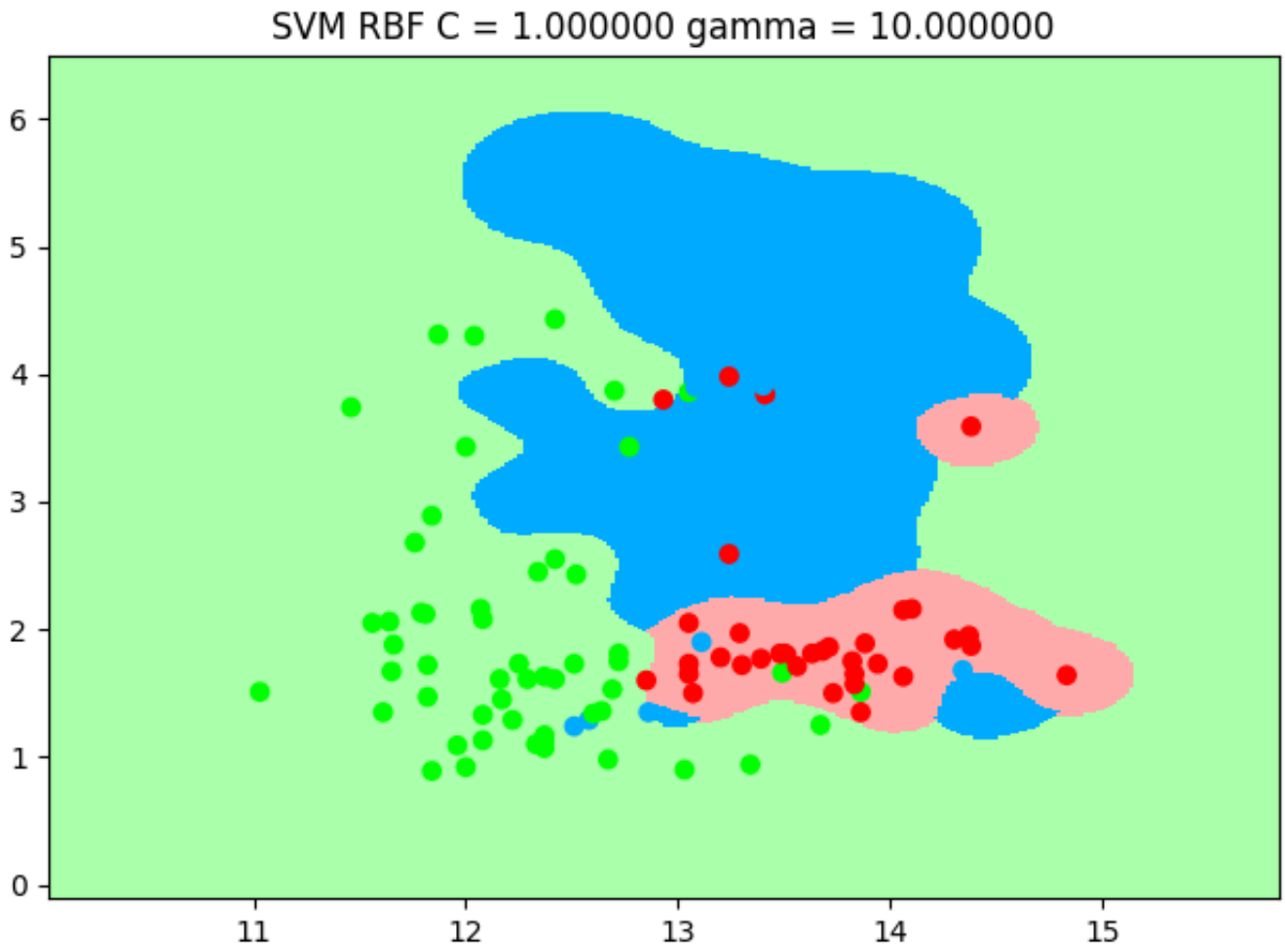
13) It's clear that C= 0.1 is the best paramiter.The final model is fitted over Xtrain+Xvalidation and the final accuracy score over test set is **0.84**

14) Different from the last models, the new ones are characterized by non linear boundaries and sometimes different regions of classification for the same class.This is due to mapping to an higher space,provided by kernel, in which a linear separation of classes is more accurate.The kernel trick is defined as k(xi,xj) = **Φ**(xi)**Φ**(xj),in which **Φ** is the mapping function,so this allow to use the dot product of the new dimensional space without mapping xi to **Φ**(xi) and it is faster.

15) To perform a better tuning between gamma and C was performed a grid seach to find the best couple of these paramiters. The code below select the couple of c and gamma that get the best accuracy score on the valudation set.The paramiter selected are gamma = **10** and C = **1**

```
C_range = 10. ** np.arange(-3, 8)
gamma_range = 10. ** np.arange(-5, 4)

max =.0
gamma_max = .0
C_max = .0
for c in C_range:
    for g in gamma_range:
        svf = svm.SVC(kernel='rbf',gamma=g,C=c)
        svf.fit(X_train[:,:2],Y_train)
        score = mc.accuracy_score(Y_validation,svf.predict(X_validation[:,:2]))
        if(score > max):
            max = score
            gamma_max = g
            C_max = c
svf = svm.SVC(kernel='rbf',gamma=g,C=c)
svf.fit(X_fold[:,:2],Y_fold)
score = mc.accuracy_score(Y_test,svf.predict(X_test[:,:2]))
```

The representation of the model is:



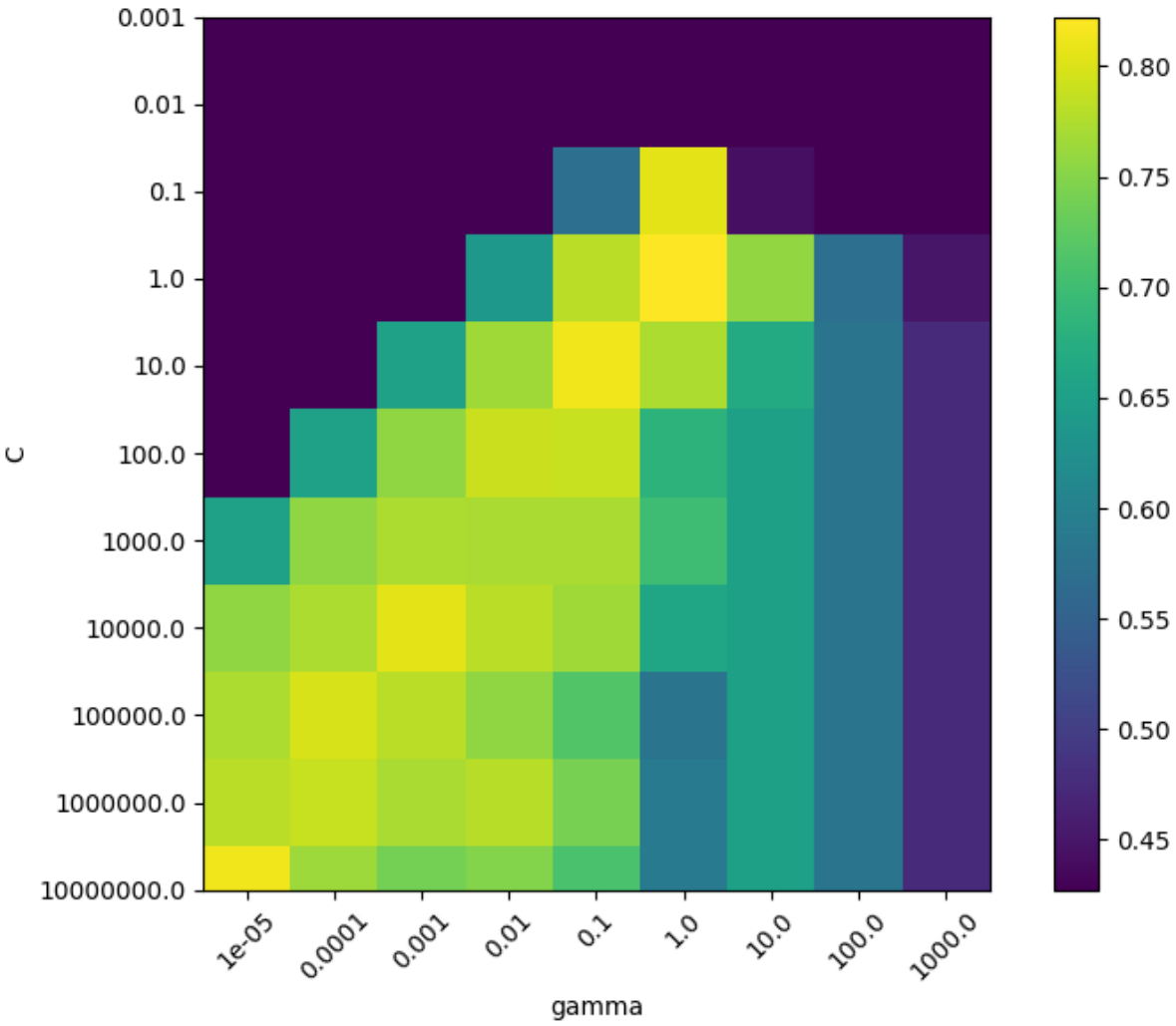
This model give an accuracy of **0.76** over test set in contrast of an accuracy of **0.83** over validation set.This

5-fold

16,17) Another tecnique to find the best paramiters C and gamma is K-fold cross validation.The follwing code performs a first K-fold over C*range and gamma*range using GridSearchCV that implements K-fold

```
X_fold = np.concatenate((X_train,X_validation))
Y_fold = np.concatenate((Y_train,Y_validation))
C_range = 10. ** np.arange(-3, 8)
gamma_range = 10. ** np.arange(-5, 4)
param_grid = dict(gamma=gamma_range, C=C_range)

grid = ms.GridSearchCV(svm.SVC(), param_grid=param_grid, cv=5,iid=False)

grid.fit(X_fold[:,:2],Y_fold)
```

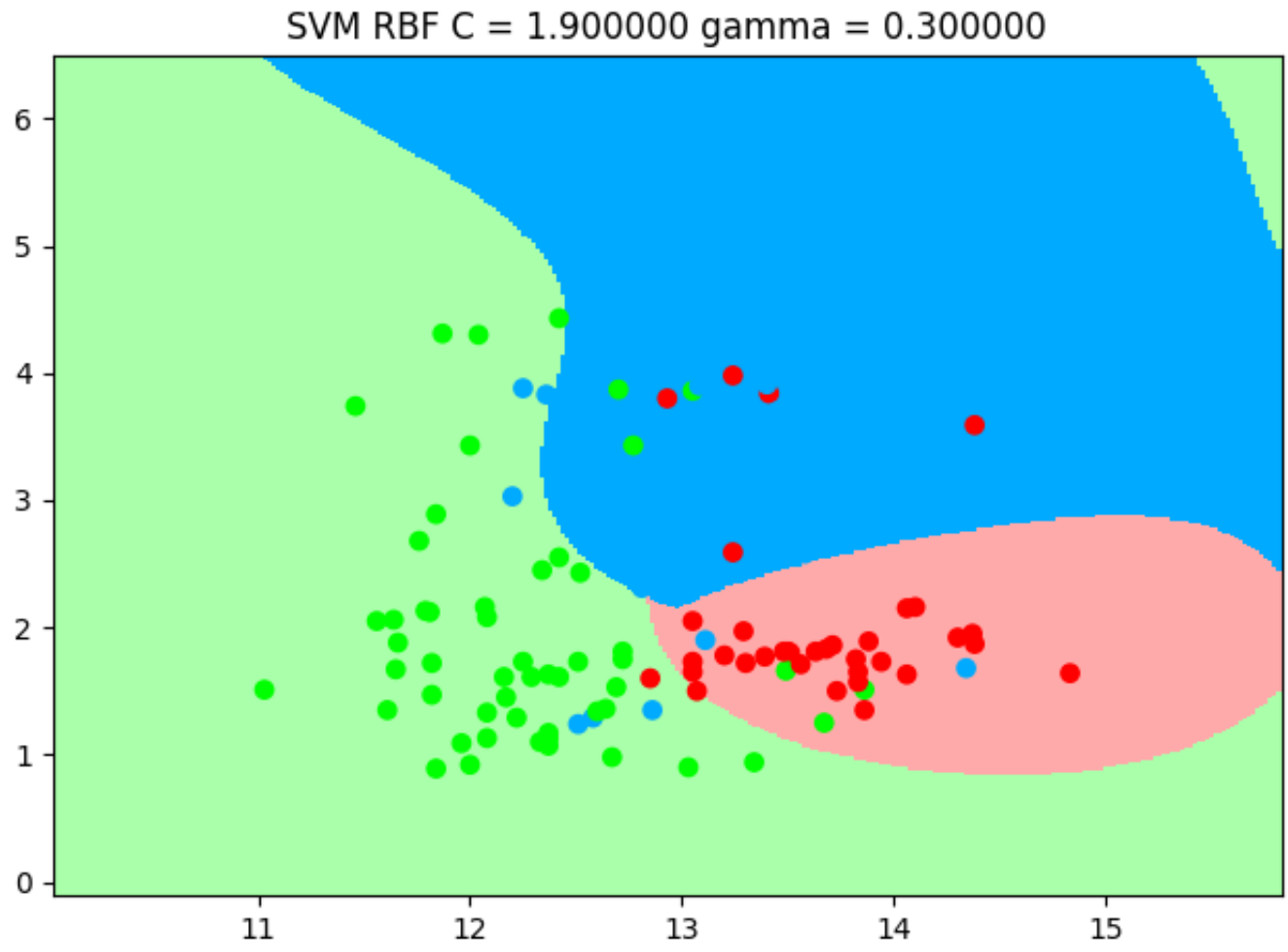
The result of this operatio is visible in this representation:



This graph represents grid search of best C and Gamma.It's visible that the best value of C is between 10 and 0.1 and the best gamma is between 0.01 and 1.To do a better tuning of C and gamma is done a K-fold over this 2 ranges with the following code:

```
gamma_range = np.linspace(grid.best_params_['gamma']/10,grid.best_params_['gamma']
*10,100)
C_range = np.linspace(grid.best_params_['C']/10,grid.best_params_['C']*10,100)
param_grid = dict(gamma=gamma_range, C=C_range)

grid = ms.GridSearchCV(svm.SVC(), param_grid=param_grid, cv=5,iid=False)

grid.fit(X_fold[:,:2],Y_fold)
model = svm.SVC(kernel='rbf',gamma=grid.best_params_['gamma'],C=grid.best_params_[
'C'])
model.fit(X_fold[:,:2],Y_fold)
printSVM(X_fold[:,:2],Y_fold,model,10,"SVM RBF C = %f gamma = %f" % (grid.best_par
ams_['C'],grid.best_params_['gamma']))
print(grid.score(X_test[:,:2],Y_test))
print(grid.best_params_)
```

18) The final model provide an accuracy on the test set equal to **0.84**.This is the graph:



This last model is better than the previus one because the k-fold select the couple paramiters with which provide the best accuracy average given score of each of the 5 splits and so provide the models that give more generalization.

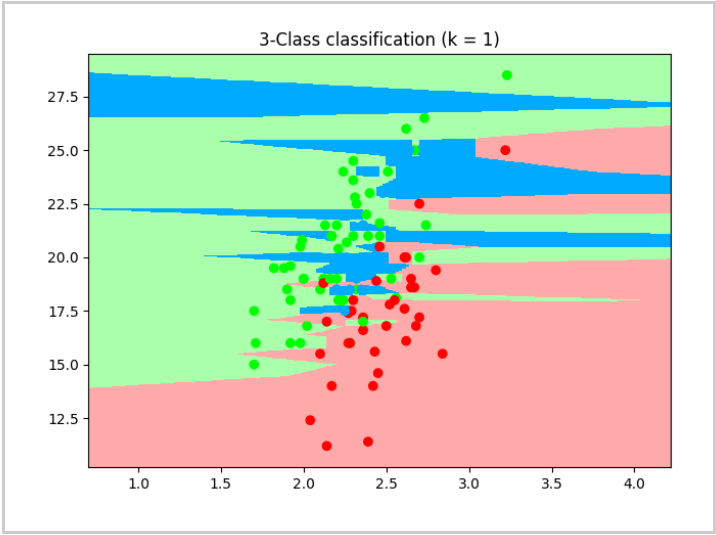
Discussion between SVM and KNN

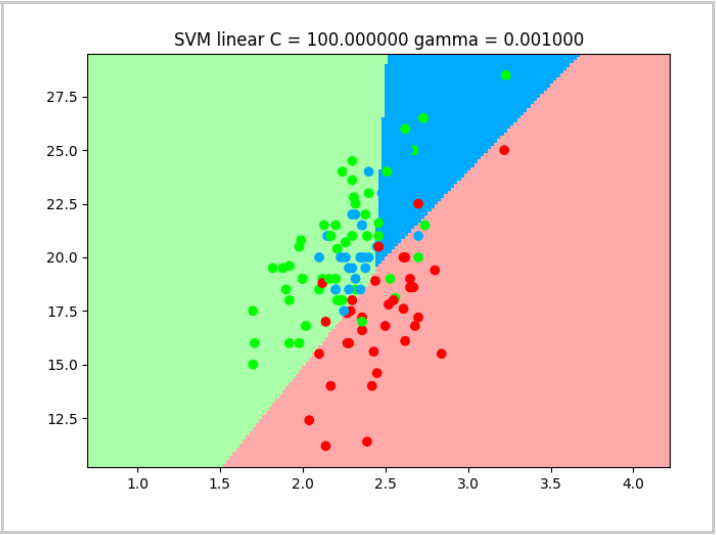
19)KNN is a very simple and intuitive algorithm that provides a non linear classifier with good classification performance.SVM is natively linear but can be non linear using kernel.It's classification performance are generally better than knn and it is computationally less expensive than knn because the final model keeps track only a few points from training set (support vectors) instead of KNN that use all training data for classification.

Models using different

20) The following models are trained using the second 2 features of the dataset.Are produced 2 models, a KNN and a SVM using k-fold fro tuning paramiters of both models.Below there are the code and the pictures of models:

```
X = load_wine()

X_train,X_test,Y_train,Y_test= train_test_split(X.data,X.target,test_size=0.3,rand
om_state=1)

gamma =[0.001,0.01,0.1,1,10,100,1000]
C = [0.001,0.01,0.1,1,10,100,1000]
k =[1,3,5,7]
paramitersSVM = [{'kernel':['rbf','linear'],'gamma':gamma,'C':C}]
paramitersKNN =[{'n_neighbors':k}]

foldKNN = ms.GridSearchCV(knn.KNeighborsClassifier(),paramitersKNN,cv=5,iid=False)
foldSVM = ms.GridSearchCV(svm.SVC(),paramitersSVM,cv=5,iid=False)

foldKNN.fit(X_train[:,2:4],Y_train)
foldSVM.fit(X_train[:,2:4],Y_train)

KNN = knn.KNeighborsClassifier(foldKNN.best_params_['n_neighbors'])
SVM = svm.SVC(kernel=foldSVM.best_params_['kernel'],C=foldSVM.best_params_['C'],ga
mma=foldSVM.best_params_['gamma'])
KNN.fit(X_train[:,2:4],Y_train)
SVM.fit(X_train[:,2:4],Y_train)
```


3-Class classification (k = 1)
27.5
25.0
22.5
20.0
17.5
15.0
12.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


SVM linear C = 100.000000 gamma = 0.001000
27.5
25.0
22.5
20.0
17.5
15.0
12.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0